暗号アルゴリズム「Hierocrypt-3」
詳細評価（HW実装評価）レポート

２００１年1月１９日

目次

１．アルゴリズム概要

Hierocrypt-3は、2000年に東芝より提案された、ブロック長128 ビット、鍵長128ビット、192ビット、256ビットとして設計された暗号アルゴリズムである。Hierocrypt-3の設計において、次の点を重視したアルゴリズムとして提案されている。

- 主要な攻撃法に対する十分な安全性
- スマートカードやミドルウェアでの暗号化の高速性
- 実装の効率性
- 設計の透明性

以上の条件を満足するために、データ攪拌部の設計にはSPN構造の一種である入れ子型SPN構造と言うものが採用されている。入れ子型SPN構造とは、上位SPN構造のS-boxの位置に下位SPN構造を埋め込んだ、再帰的階層構造となっている。また、鍵スケジュール部の設計にはFeistel構造が利用されている。

Hierocrypt-3は、128 ビットブロックでSPN構造の暗号であるSQUAREやRijndaelで攻撃対象となっているSQUARE攻撃により強い暗号の設計を目指し、入れ子型SPN 構造が採用されている。

２．アルゴリズムHierocrypt-3の説明

２．１　データランダマイズ部

図 1、図 2 はそれぞれ Hierocrypt-3 のデータランダマイズ部の暗号化部と復号化部を表している。一般的な Feistel 構造の場合は、暗号化と復号化において、共通のデータランダマイズ部を利用することが可能であるが、SPN 構造を持つ Hierocrypt-3 では、暗号化と復号化で別々の構成を持たなければならない。

暗号化部は、鍵サイズが 128 ビットの場合、図１のように、段関数ρを 5 段繰り返し、その後、XS 関数を 1 段実行し、最後に 128 ビットの拡大鍵 K^7 と排他的論理和するアルゴリズムである。復号化部は図 2 のように、最初に 128 ビットの拡大鍵 K^7 と排他的論理和を行い、XS 関数の逆関数である XS^{-1} 関数を 1 段実行し、段関数ρの逆関数である段関数 ρ^{-1} を 5 段繰り返すアルゴリズムである。

なお、鍵サイズが 192 ビットの場合は、暗号化部で段関数ρを 6 段、復号化部で段関数 ρ^{-1} を 6 段処理し、鍵サイズが 256 ビットの場合は、暗号化部で段関数ρを 7 段、復号化部で段関数 ρ^{-1} を 7 段処理し

暗号化部で使用される段関数ρは、128 ビットの入力データと 128 ビットの拡大鍵 K_1^n（ｎは段数）を排他的論理和し、その結果を上位から 8 ビット毎に区切って 16 個の関数ｓに入

力する。関数ｓから出力されたデータを上位から 32 ビットずつ４つに分割し、それぞれｍｄｓ$_L$関数で処理する。ｍｄｓ$_L$関数から出力されたデータを 128 ビットデータに連接し、128 ビットの拡大鍵 K_2^n(ｎは段数)を排他的論理和し、その結果を、また、上位から 8 ビット毎に区切って 16 個の関数ｓに入力する。関数ｓから出力されたデータをＭＤＳ$_H$関数で処理を行い、段関数ρの 128 ビットの出力データとなる。

暗号化部で最後の 1 段に使用される XS 関数は、段関数ρの処理から最後のＭＤＳ$_H$関数の処理を省いたものである。

復号化部で使用される段関数ρ^{-1}は、最初にＭＤＳ$_H$関数の逆関数であるＭＤＳ$_H^{-1}$関数で処理を行い、上位から 8 ビット毎に区切って 16 個の関数ｓの逆関数である関数ｓ$^{-1}$に入力する。関数ｓ$^{-1}$から出力されたデータを 128 ビットに連接し、128 ビットの拡大鍵 K_2^n（ｎは段数）を排他的論理和する。その結果を上位から 32 ビットずつ４つに分割し、それぞれｍｄｓ$_L$関数の逆関数であるｍｄｓ$_L^{-1}$関数で処理し、上位から 8 ビット毎に区切って 16 個の関数ｓの逆関数である関数ｓ$^{-1}$に入力する。関数ｓ$^{-1}$から出力されたデータを 128 ビットに連接し、128 ビットの拡大鍵 K_1^n（ｎは段数）を排他的論理和したものが段関数ρ^{-1}の 128 ビットの出力データとなる。

復号化部で最初の 1 段に使用される XS^{-1}関数は、段関数ρ^{-1}の処理から最初のＭＤＳ$_H^{-1}$関数の処理を省いたものである。

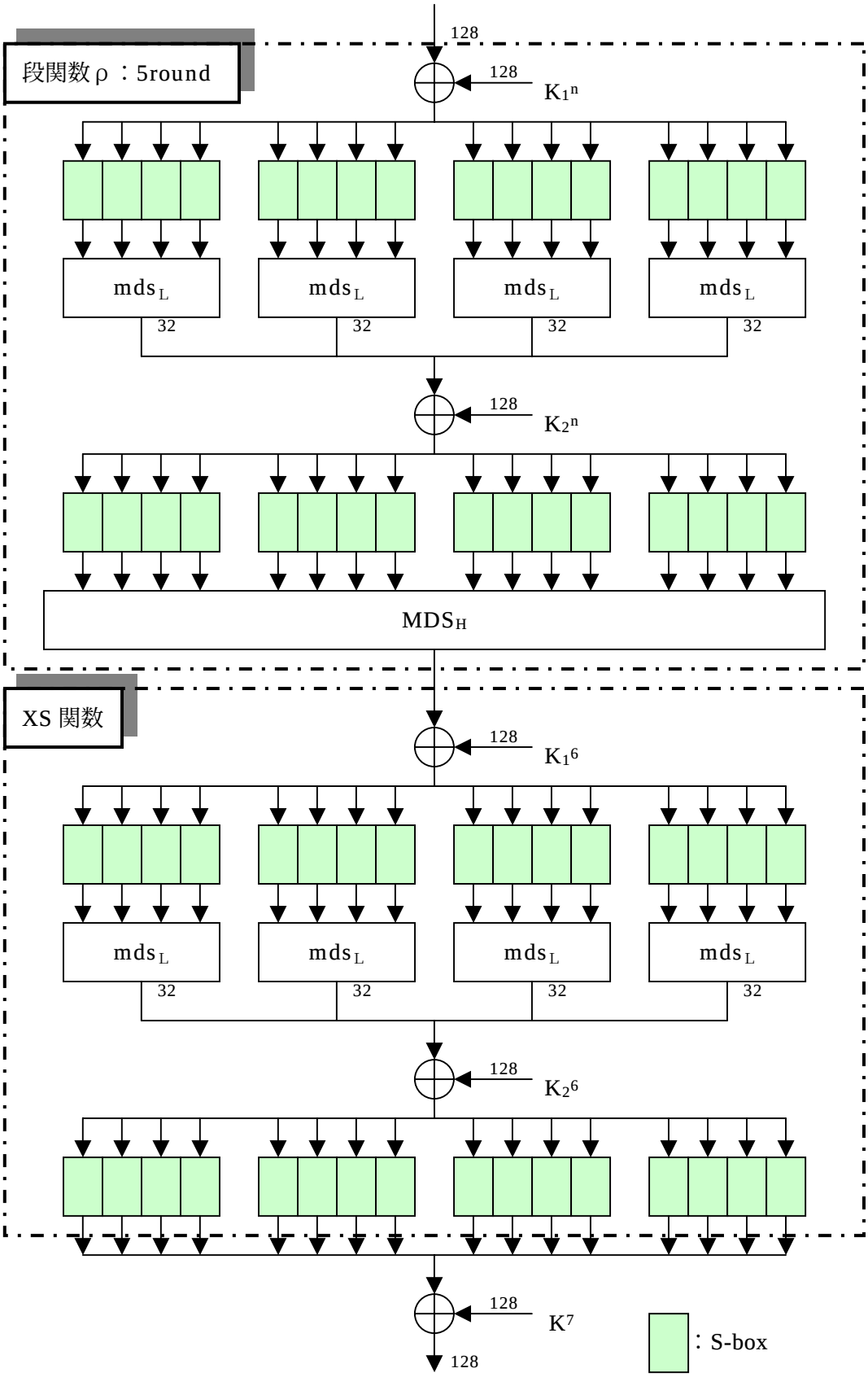


図1．データランダマイズ部(暗号化)

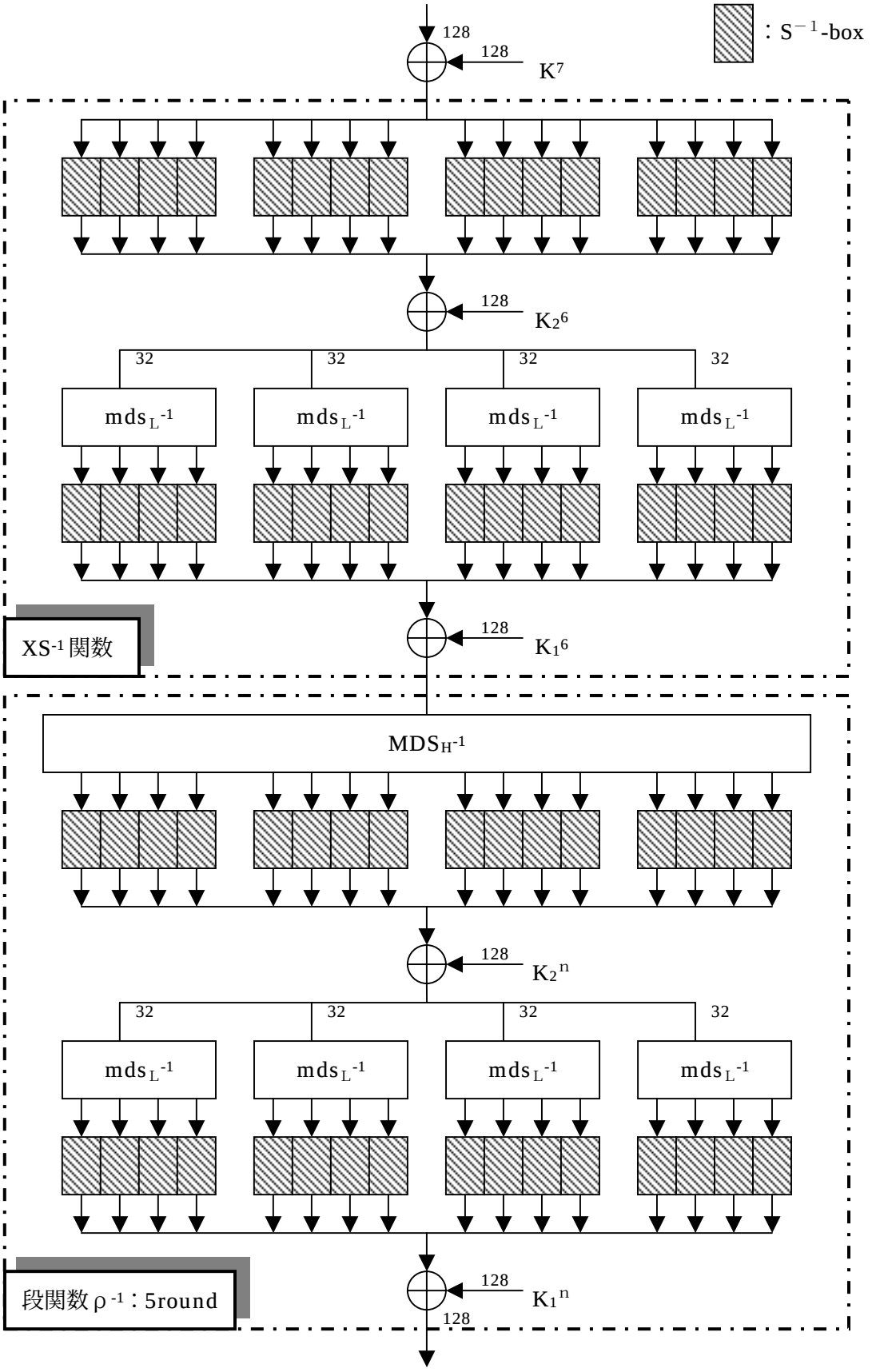


図2．データランダマイズ部(復号化)

暗号化部で使用されるＭＤＳ$_H$ 関数は以下のようなものである。x1(8)、y1(8)等は､8ビットデータを表している。

$$
\begin{pmatrix} y\,1(8) \\ y\,2(8) \\ y\,3(8) \\ y\,4(8) \\ y\,5(8) \\ y\,6(8) \\ y\,7(8) \\ y\,8(8) \\ y\,9(8) \\ y10(8) \\ y11(8) \\ y12(8) \\ y13(8) \\ y14(8) \\ y15(8) \\ y16(8) \end{pmatrix}
=
\begin{pmatrix}
1&0&1&0&1&0&1&0&1&1&0&1&1&1&1&1 \\
1&1&0&1&1&1&0&1&1&1&1&0&0&1&1&1 \\
1&1&1&0&1&1&1&0&1&1&1&1&0&0&1&1 \\
0&1&0&1&0&1&0&1&1&0&1&0&1&1&1&0 \\
1&1&1&1&1&0&1&0&1&0&1&0&1&1&0&1 \\
0&1&1&1&1&1&0&1&1&1&0&1&1&1&1&0 \\
0&0&1&1&1&1&1&0&1&1&1&0&1&1&1&1 \\
1&1&1&0&0&1&0&1&0&1&0&1&1&0&1&0 \\
1&1&0&1&1&1&1&1&1&0&1&0&1&0&1&0 \\
1&1&1&0&0&1&1&1&1&1&0&1&1&1&0&1 \\
1&1&1&1&0&0&1&1&1&1&1&0&1&1&1&0 \\
1&0&1&0&1&1&1&0&0&1&0&1&0&1&0&1 \\
1&0&1&0&1&1&0&1&1&1&1&1&1&0&1&0 \\
1&1&0&1&1&1&1&0&0&1&1&1&1&1&0&1 \\
1&1&1&0&1&1&1&1&0&0&1&1&1&1&1&0 \\
0&1&0&1&1&0&1&0&1&1&1&0&0&1&0&1
\end{pmatrix}
\begin{pmatrix} x\,1(8) \\ x\,2(8) \\ x\,3(8) \\ x\,4(8) \\ x\,5(8) \\ x\,6(8) \\ x\,7(8) \\ x\,8(8) \\ x\,9(8) \\ x10(8) \\ x11(8) \\ x12(8) \\ x13(8) \\ x14(8) \\ x15(8) \\ x16(8) \end{pmatrix}
$$

y1 = x1 + x3 + x5 + x7 + x9 + x10 + x12 + x13 + x14 + x15 + x16
y2 = x1 + x2 + x4 + x5 + x6 + x8 + x9 + x10 + x11 + x14 + x15 + x16
y3 = x1 + x2 + x3 + x5 + x6 + x7 + x9 + x10 + x11 + x15 + x16
y4 = x2 + x4 + x6 + x8 + x9 + x11 + x13 + x14 + x15
y5 = x1 + x2 + x3 + x4 + x5 + x7 + x9 + x11 + x13 + x14 + x16
y6 = x2 + x3 + x4 + x5 + x6 + x8 + x9 + x10 + x12 + x13 + x14 + x15
y7 = x3 + x4 + x5 + x6 + x7 + x9 + x10 + x11 + x13 + x14 + x15 + x16
y8 = x1 + x2 + x3 + x6 + x8 + x10 + x12 + x13 + x15
y9 = x1 + x2 + x4 + x5 + x6 + x7 + x8 + x9 + x11 + x13 + x15
y10 = x1 + x2 + x3 + x6 + x7 + x8 + x9 + x10 + x12 + x13 + x14 + x16
y11 = x1 + x2 + x3 + x4 + x7 + x8 + x9 + x10 + x11 + x13 + x14 + x15
y12 = x1 + x3 + x5 + x6 + x7 + x10 + x12 + x14 + x16
y13 = x1 + x3 + x5 + x6 + x8 + x9 + x10 + x11 + x12 + x13 + x15
y14 = x1 + x2 + x4 + x5 + x6 + x7 + x10 + x11 + x12 + x13 + x14 + x16
y15 = x1 + x2 + x3 + x5 + x6 + x7 + x8 + x11 + x12 + x13 + x14 + x15
y16 = x2 + x4 + x5 + x7 + x9 + x10 + x11 + x14 + x16

復号化部で使用されるMDS_H^{-1}関数は、上記行列の逆行列であるが、Hierocrypt-L1の仕様書には記載されていないため、割愛する。

暗号化部で使用されるmds_L関数は以下のようなものである。x1(8)、y1(8)等は、8ビットデータを表している。

$$\begin{pmatrix} y1(8) \\ y2(8) \\ y3(8) \\ y4(8) \end{pmatrix} = \begin{pmatrix} C4 & 65 & C8 & 8B \\ 8B & C4 & 65 & C8 \\ C8 & 8B & C4 & 65 \\ 65 & C8 & 8B & C4 \end{pmatrix} \begin{pmatrix} x1(8) \\ x2(8) \\ x3(8) \\ x4(8) \end{pmatrix}$$

$$y1 = C4*x1 + 65*x2 + C8*x3 + 8B*x4$$
$$y2 = 8B*x1 + C4*x2 + 65*x3 + C8*x4$$
$$y3 = C8*x1 + 8B*x2 + C4*x3 + 65*x4$$
$$y4 = 65*x1 + C8*x2 + 8B*x3 + C4*x4$$

また、利用されている原始多項式は以下のようなものである。
$GF(2^8)$の原始多項式：$p(z) = z^8 + z^6 + z^5 + z + 1$

復号化部で使用されるmds_L^{-1}関数は以下のようなものである。x1(8)、y1(8)等は、8ビットデータを表している。

$$\begin{pmatrix} x1(8) \\ x2(8) \\ x3(8) \\ x4(8) \end{pmatrix} = \begin{pmatrix} 82 & C4 & 34 & F6 \\ F6 & 82 & C4 & 34 \\ 34 & F6 & 82 & C4 \\ C4 & 34 & F6 & 82 \end{pmatrix} \begin{pmatrix} y1(8) \\ y2(8) \\ y3(8) \\ y4(8) \end{pmatrix}$$

$$x1 = 82*y1 + C4*y2 + 34*y3 + F6*y4$$
$$x2 = F6*y1 + 82*y2 + C4*y3 + 34*y4$$
$$x3 = 34*y1 + F6*y2 + 82*y3 + C4*y4$$
$$x4 = C4*y1 + 34*y2 + F6*y3 + 82*y4$$

暗号化部で使用される関数ｓは以下のようなものである。

関数ｓ ＝ ｛

07 FC 55 70 98 8E 84 4E BC 75 CE 18 02 E9 5D 80
1C 60 78 42 9D 2E F5 E8 C6 7A 2F A4 B2 5F 19 87
0B 9B 9C D3 C3 77 3D 6F B9 2D 4D F7 8C A7 AC 17
3C 5A 41 C9 29 ED DE 27 69 30 72 A8 95 3E F9 D8
21 8B 44 D7 11 0D 48 FD 6A 01 57 E5 BD 85 EC 1E
37 9F B5 9A 7C 09 F1 B1 94 81 82 08 FB C0 51 0F
61 7F 1A 56 96 13 C1 67 99 03 5E B6 CA FA 9E DF
D6 83 CC A2 12 23 B7 65 D0 39 7D 3B D5 B0 AF 1F
06 C8 34 C5 1B 79 4B 66 BF 88 4A C4 EF 58 3F 0A
2C 73 D1 F8 6B E6 20 B8 22 43 B3 33 E7 F0 71 7E
52 89 47 63 0E 6D E3 BE 59 64 EE F6 38 5C F4 5B
49 D4 E0 F3 BB 54 26 2B 00 86 90 FF FE A6 7B 05
AD 68 A1 10 EB C7 E2 F2 46 8A 6C 14 6E CF 35 45
50 D2 92 74 93 E1 DA AE A9 53 E4 40 CD BA 97 A3
91 31 25 76 36 32 28 3A 24 4C DB D9 8D DC 62 2A
EA 15 DD C2 A5 0C 04 1D 8F CB B4 4F 16 AB AA A0
｝;

復号化部で使用される関数s^{-1}は、上記行列の逆行列であるが、Hierocrypt-L1の仕様書には記載されていないため、割愛する。

２．２　鍵スケジュール部

Hierocrypt-3の鍵スケジュール部は、図3のような繰り返し段構造から成る中間鍵生成部と、各段の中間鍵から拡大鍵を生成する拡大鍵生成部から構成されている。なお、ここでは、鍵サイズが128ビットの場合について記述を行う。

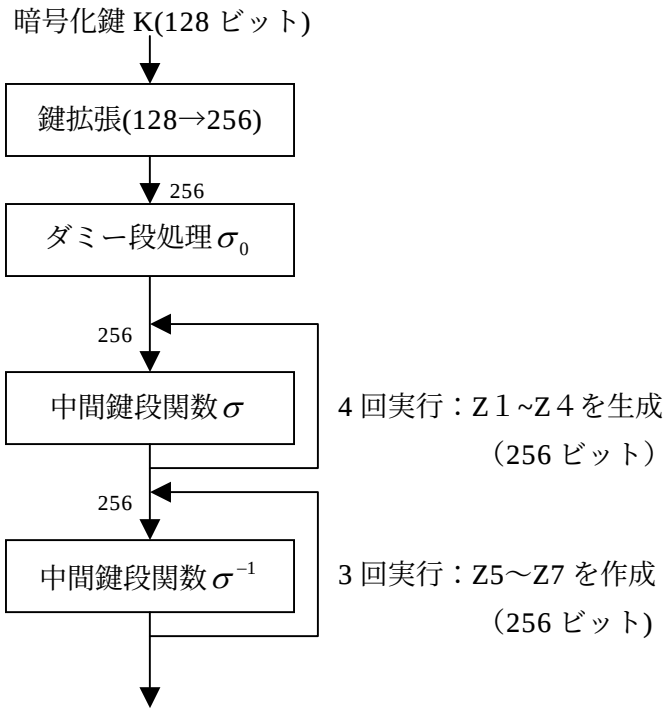
中間鍵生成部は、図3のように128ビットの暗号化鍵Kを256ビットへ鍵拡張し、ダミー段処理σ_0で処理を行い、その後、中間鍵段関数σを4回、中間鍵段関数σ^{-1}を3回実行して、128ビットの中間鍵Z1からZ7を作成します。また、Hierocrypt-3の鍵スケジュール部では、以下のような32ビットの段依存定数を使用するので、定義しておく。32ビットの定数を2つずつ組み合わせて、鍵スケジュール部内で使用する。

H0 = 0x5A827999

H1 = 0x6ED9EBA1

H2 = 0x8F1BBCDC

H3 = 0xCA62C1D6



※　Z5=Z4 , Z6=Z3 , Z7=Z2　が成立する。

図３．鍵スケジュール部(中間鍵生成部)

中間鍵生成部の最初の処置である鍵拡張は図 4 のように、入力された 128 ビットの鍵を上記で定義した H3 と H2 を連接した 64 ビットデータを最後に付加した形の 256 ビットデータを出力するように処理される。

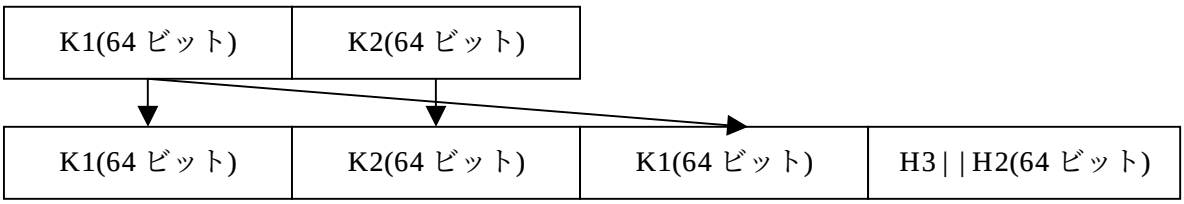


図 4. 鍵拡張

中間鍵生成部で使用するダミー段処理σ_0は、図 5 のように 256 ビット入力 256 ビット出力の関数である。256 ビットの入力を 64 ビット毎に分割し、2 回の M_{5E} 関数、F_σ関数と 3 回の排他的論理和を処理する。

中間鍵生成部で使用する中間鍵段関数σは、図 6 のように 256 ビット入力 256 ビット出力の関数である。256 ビットの入力を 64 ビット毎に分割し、$P^{(32)}$関数、2 回の M_{5E} 関数、F_σ関数と 3 回の排他的論理和を処理する。

中間鍵生成部で使用する中間鍵段関数σ^{-1}は、図 7 のように 256 ビット入力 384 ビット出力（W1 と W2 も拡大鍵生成部で使用するため）の関数である。256 ビットの入力を 64 ビット毎に分割し、$P^{(32)-1}$関数、2 回の M_{B3} 関数、F_σ関数と 3 回の排他的論理和を処理する。

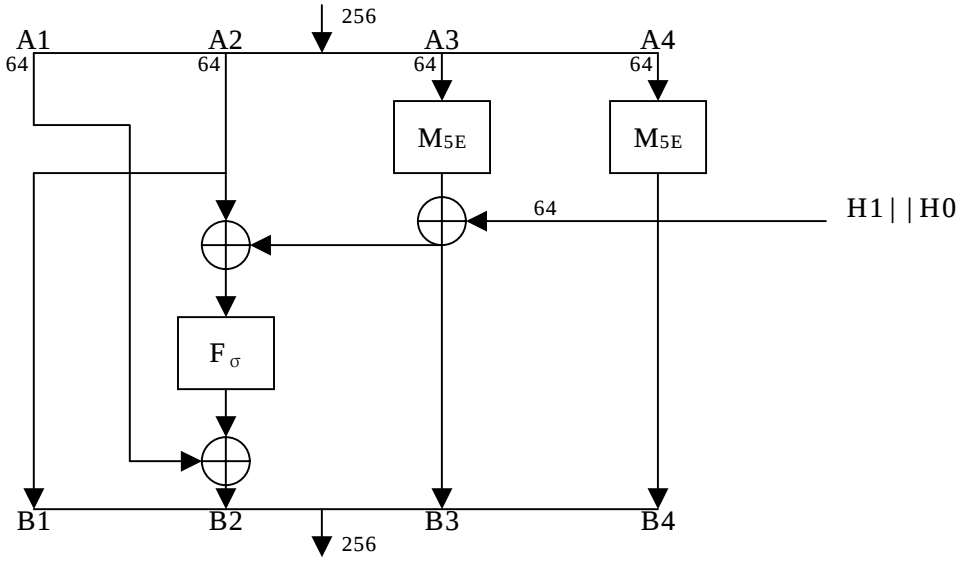


図 5. ダミー段処理σ_0

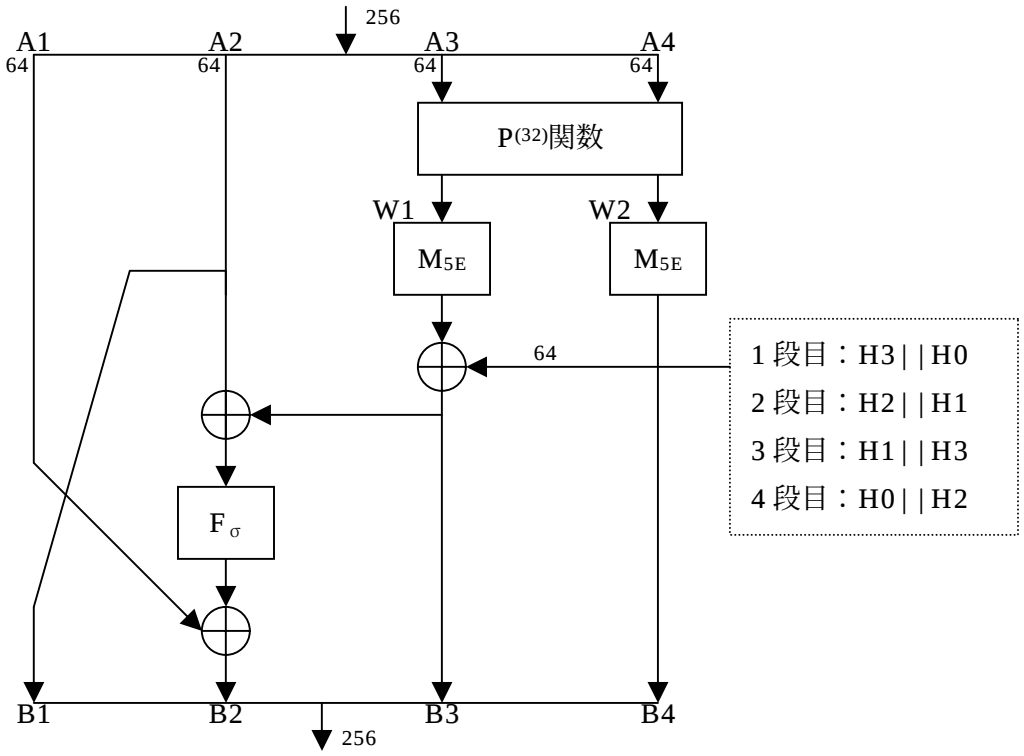


図 6. 中間鍵段関数σ

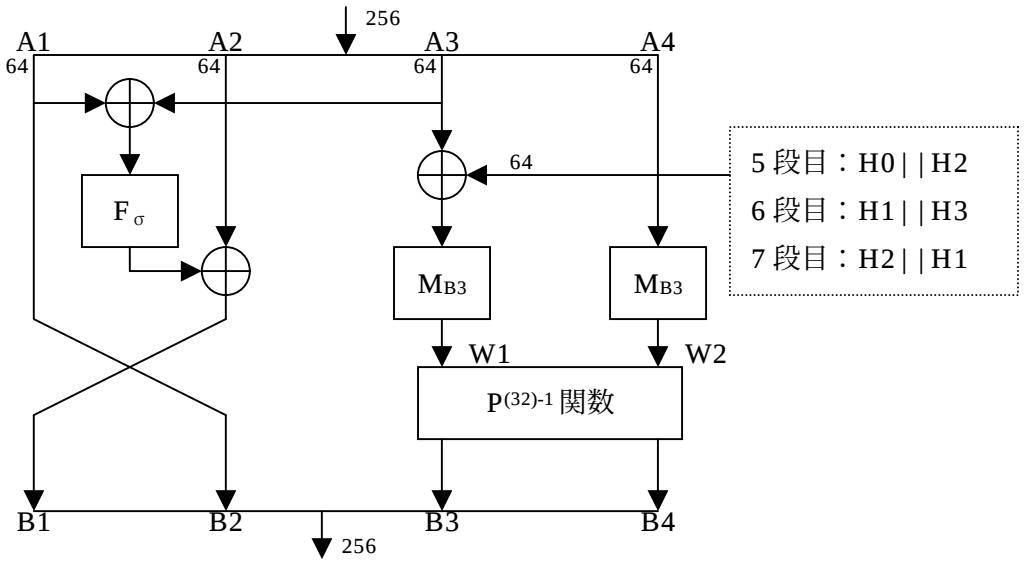


図 7. 中間鍵段関数σ^{-1}

中間鍵生成部で使用するP(n)関数および、P(n)-1 関数は以下のようなものである。x1(n)、y1(n)等は、nビットデータを表している。

P(n) 関数

$$\begin{pmatrix} y1(n) \\ y2(n) \\ y3(n) \\ y4(n) \end{pmatrix} = \begin{pmatrix} 1\ 0\ 1\ 0 \\ 0\ 1\ 0\ 1 \\ 0\ 1\ 1\ 1 \\ 1\ 0\ 1\ 1 \end{pmatrix} \begin{pmatrix} x1(n) \\ x2(n) \\ x3(n) \\ x4(n) \end{pmatrix}$$

P(n)-1 関数

$$\begin{pmatrix} x1(n) \\ x2(n) \\ x3(n) \\ x4(n) \end{pmatrix} = \begin{pmatrix} 1\ 1\ 1\ 0 \\ 1\ 1\ 0\ 1 \\ 0\ 1\ 1\ 0 \\ 1\ 0\ 0\ 1 \end{pmatrix} \begin{pmatrix} y1(n) \\ y2(n) \\ y3(n) \\ y4(n) \end{pmatrix}$$

中間鍵生成部で使用するM_{5E} 関数およびM_{B3} 関数は以下のようなものである。x1(8)、y1(8)等は、8ビットデータを表している。

M_{5E} 関数

$$\begin{pmatrix} y1(8) \\ y2(8) \\ y3(8) \\ y4(8) \end{pmatrix} = \begin{pmatrix} 1\ 0\ 1\ 0 \\ 1\ 1\ 0\ 1 \\ 1\ 1\ 1\ 0 \\ 0\ 1\ 0\ 1 \end{pmatrix} \begin{pmatrix} x1(8) \\ x2(8) \\ x3(8) \\ x4(8) \end{pmatrix}$$

$$\begin{pmatrix} y5(8) \\ y6(8) \\ y7(8) \\ y8(8) \end{pmatrix} = \begin{pmatrix} 1\ 1\ 1\ 1 \\ 0\ 1\ 1\ 1 \\ 0\ 0\ 1\ 1 \\ 1\ 1\ 1\ 0 \end{pmatrix} \begin{pmatrix} x5(8) \\ x6(8) \\ x7(8) \\ x8(8) \end{pmatrix}$$

M_{B3} 関数

$$\begin{pmatrix} y1(8) \\ y2(8) \\ y3(8) \\ y4(8) \end{pmatrix} = \begin{pmatrix} 0\ 1\ 0\ 1 \\ 1\ 0\ 1\ 0 \\ 1\ 1\ 0\ 1 \\ 1\ 0\ 1\ 1 \end{pmatrix} \begin{pmatrix} x1(8) \\ x2(8) \\ x3(8) \\ x4(8) \end{pmatrix}$$

$$\begin{pmatrix} y5(8) \\ y6(8) \\ y7(8) \\ y8(8) \end{pmatrix} = \begin{pmatrix} 1\ 1\ 0\ 0 \\ 0\ 1\ 1\ 0 \\ 1\ 0\ 1\ 1 \\ 1\ 0\ 0\ 1 \end{pmatrix} \begin{pmatrix} x5(8) \\ x6(8) \\ x7(8) \\ x8(8) \end{pmatrix}$$

中間鍵生成部で使用するF_σ関数は図8のように64ビット入力64ビット出力の関数である。64ビットの入力を8ビット毎に分割し8つの関数sと、P(16)関数を処理する。

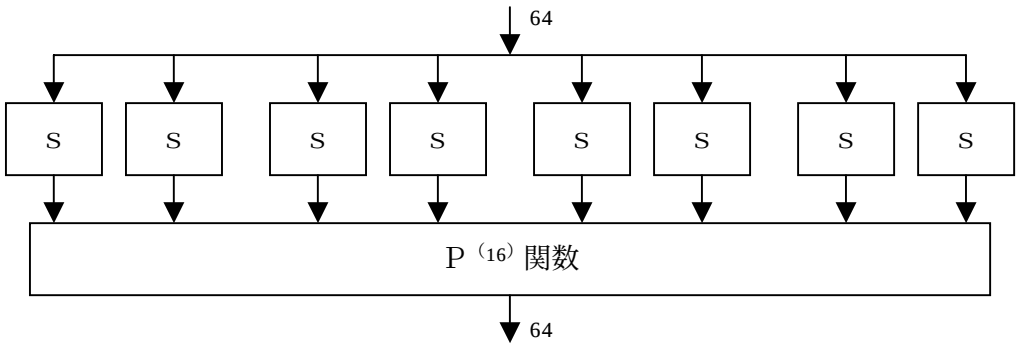


図 7. F_σ関数

鍵スケジュール部の拡大鍵生成部に関しては、下記の式により、ｔ段目の64ビット毎の鍵$K^{(t)}_1$から$K^{(t)}_4$までが生成される。１段目から4段目までは拡大鍵生成(平文側)を処理し、5段目から7段目までは拡大鍵生成(暗号文側)を処理する。

なお、ｔ＝７の時の$K^{(t)}_3$ 、$K^{(t)}_4$ は使用しないことに注意する。下記の式に使用されているF_σ関数は、中間鍵生成部の使用した関数を利用している。

[拡大鍵生成(平文側)(1 ≦ t ≦ 4)：それぞれの処理は64ビット

$V^{(t)} = F_\sigma \quad (Z^{(t-1)}_2 \quad + \quad Z^{(t)}_3)$

$K^{(t)}_1 = Z^{(t-1)}_1 + V^{(t)}$

$K^{(t)}_2 = Z^{(t)}_3 \quad + V^{(t)}$

$K^{(t)}_3 = Z^{(t)}_4 \quad + V^{(t)}$

$K^{(t)}_4 = Z^{(t-1)}_2 + Z^{(t)}_4$

[拡大鍵生成(暗号文側)(5 ≦ t ≦ 7) ：それぞれの処理は64ビット

$V^{(t)} = F_\sigma \quad (Z^{(t-1)}_1 \quad + \quad Z^{(t-1)}_3)$

$K^{(t)}_1 = Z^{(t)}_1 \quad + Z^{(t-1)}_3$

$K^{(t)}_2 = W^{(t)}_1 \quad + V^{(t)}$

$K^{(t)}_3 = W^{(t)}_2 \quad + V^{(t)}$

$K^{(t)}_4 = Z^{(t-1)}_1 + W^{(t)}_2$

３．評価方針

今回の性能評価は、SBOX、乗算器、加算器、ラウンド関数等といった基本的な機能を実現している個々のパーツ(以下、primitive)の評価(回路規模、処理速度等)を実施することによりアルゴリズム全体の性能を見積る方法で評価を行い、また、個々の primitive においては、実際の LSI 作成に則した条件を付加して評価を行うことを前提に行った。なお、本報告で用いる評価環境は、我々が H/W 評価経験のある三菱 0.35μm CMOS ASIC ライブラリを用い、回路記述には Verilog-HDL、Synthesis には Design Compiler を用い、回路規模(ゲート数)およびクリティカルパス長、処理速度等の性能見積を行った。

３.１　回路規模・性能の見積方法

理想的と考えられる評価は、個々のアルゴリズムに対し、公平な評価指標のもとに、最大限の optimize（SBOX の小型化および最速化、および SBOX やラウンド関数のような複数回使用される回路の実装個数と繰り返し回数のトレードオフ等の検討）を試みて回路規模、性能を見積ることである。しかし、この評価方法では、開発時間、開発資源（論理合成ツール等のソフトウェア、コンピュータ等のハードウェアおよび開発者等）が大量に必要となり、本詳細評価期間では、全てのアルゴリズムを公平な指標のもとに optimize することは困難である。そこで特定のアルゴリズムだけを最適化することはせずに、アルゴリズムを全て実装（回路規模は大きくても構わない）し、クリティカルパス長の短縮（処理速度向上）を重視して評価を行った。

そこで今回の評価方法は、以下の通りに行った。

① 原則として、評価対象アルゴリズム全体(鍵スケジュールおよびすべての内部鍵レジスタを含む)をすべて H/W 実装することを前提とする。
② 入力鍵サイズは 128bit とし、性能評価時に、暗号化、復号部分と拡大鍵生成部分の分離を容易にするため、拡大鍵は全てレジスタに格納する。
③ SBOX、乗算器、加算器、ラウンド関数等といった基本的な機能を実現している個々のパーツ(以下、primitive)の評価(回路規模、処理速度等)を実施する。
④ primitive の使用個数の総和を求め回路規模とする。
⑤ クリティカルパス上に存在する primitive の遅延値すべての総和を求めクリティカルパス長とする。

⑥ 「処理速度」を以下のように定義する。

$$\mathrm{Throughput\ [Mbps]} = \frac{\text{ブロックサイズ（128[bit]）}}{\{\text{暗号化（または復号）回路のクリティカルパス [ns]}\}}$$

⑦ テーブル参照型 SBOX の実装は、論理合成ツール（Design Compiler）のみを用い optimize を行う。

⑧ 我々がハードウェア評価経験のあるツールを用いる。

・開発言語：Verilog-HDL

・シミュレータ：Verilog-XL

・デザインライブラリ：三菱 0.35μm CMOS ASIC ライブラリ

・ 論理合成および性能評価：Design Compiler（version 1998.08）

⑨ 評価するときの環境条件は、Worst ケースとする。

※注…Worst ケースとは、ハードウェアが動作する環境が劣悪、かつ、LSI 製造時のバラツキ度合いが悪い方の状態にあることを意味する。一般的に、ハードウェアの性能を議論するときには、議論の対象となっているハードウェアの動作を保証するという意味で、Worst ケースで性能を論じることがほとんどである。よって、これ以降は、何も断りが無い場合は全て Worst ケースで議論を行う。

3.2 設計上の留意点

primitive の回路設計は、Verilog-HDL で記述し、設計する際の細かなブロック分けは、可能な限りアルゴリズム開発者のブロック分けに準じるように留意し行った。

3.3 Synthesis（論理合成）条件

Synthesis(論理合成)ツールは Synopsys 社の Design Compiler(version 1998-08)を用い、すべてのアルゴリズムおよび primitive に同一の条件を付加した。この条件は、速度を最大、かつ、実際の LSI 作成に則するようなものである。以下に条件を示す。

① 加算・減算・乗算回路は、Synopsys の Design Ware Basic Library 内にある最速なライブラリを用いる。

・ 32bit 加算器は Synopsys の Design Ware Basic Library 内にある DW01_add(cla)[Carry look-ahead synthesis model]を用いる。

・ 32bit 減算器は Synopsys の Design Ware Basic Library 内にある

DW01_sub(cla)[Carry look-ahead synthesis model]を用いる。

・ 32bit 乗算器は Synopsys の Design Ware Basic Library 内にある DW02_mult(csa)[Carry-save array synthesis model]を用いる。

② その他の条件は、基本的に default 値を用いる。ただし、fanout に関する条件は以下の理由で付加する。

○fanout 条件を全く付加しないことによる問題点

ある primitive の入力端子に fanout 条件が付けられていないとその入力端子は駆動能力が「∞」のドライバより駆動されていることになり、『primitive レベルでの評価からアルゴリズム全体を試算し評価する結果』と『アルゴリズム全体のレベルでの評価する結果』でかなりの違いが出てきてしまう。

実際の LSI 設計では、各 primitive の負荷状況(fanout 等)やドライバの駆動能力は、キャラクタライズ（characterize）という手法を用い、各入出力ピンの負荷状況を自動的に設定する方法、もしくは、各入出力端子に fanout 条件を付加する方法等を用いて LSI を設計、作成している。そこで、今回は、アルゴリズム全てを実装せずに、primitive レベルから性能を見積る方法を行うため、characterize 手法は使用せず、以下のように入出力端子に fanout 設定を行う。

① primitive の入力ピンの fanin が 1 になるように条件をつける。

② primitive の出力から最大 40 個の primitive が並列に接続されても耐えうるように、primitive の出力ピンの fanout は 40 になるように条件をつける。

４．各 primitive の見積り

評価方法に従って評価を実行するために、Hierocrypt-3 の構造を詳細にチェックし、各 primitive の見積りを行った。ここでは、回路規模や速度に関しては考慮せずに、論理的にどのような構成のものがいくつあるかと言うことについて見積もりを行っている。

４．１　データランダマイズ部

表 1 にあるように、暗号化部に関しては、ρ関数が 5 個、XS 関数が 1 個の 6 段構成となっている。その他には、最後の部分の拡大鍵との 128 ビット排他的論理和（EXOR）が 1 個存在している。復号化部しても、暗号化部同様に、ρ^{-1}関数が 5 個、XS^{-1}関数が 1 個の 6 段構成となっている。その他には、最後の部分の拡大鍵との 128 ビット排他的論理和（EXOR）が 1 個存在している。

それぞれ、ρ関数、XS 関数と ρ^{-1}関数、XS^{-1}関数の構成に関しても、ほぼ同程度の primitive 構成となっていることが判る。ただし、Feistel 構造の場合と異なり、暗号化部と復号化部において、共通のデータランダマイズ部を利用することが不可能であることから、回路規模が約 2 倍ことに留意する必要がある。

表 1．データランダマイズ部の各 primitive の見積り

暗号化部	ρ関数	5 個
	XS 関数	1 個
	128 ビット EXOR	1 個

復号化部	ρ^{-1}関数	5 個
	XS^{-1}関数	1 個
	128 ビット EXOR	1 個

ρ関数	128 ビット EXOR	2 個
	MDS_H関数	1 個
	mds_L関数	4 個
	関数 s	32 個

ρ^{-1}関数	128 ビット EXOR	2 個
	MDS_H^{-1}関数	1 個
	mds_L^{-1}関数	4 個
	関数 s^{-1}	32 個

XS 関数	128 ビット EXOR	2 個
	mds_L関数	4 個
	関数 s	32 個

XS^{-1}関数	128 ビット EXOR	2 個
	mds_L^{-1}関数	4 個
	関数 s^{-1}	32 個

４．２　鍵スケジュール部

表 2 にあるように、中間鍵生成部に関しては、ダミー段処理 σ^0 が 1 個、中間鍵段関数 σ が 4 個、中間鍵段関数 σ^{-1} が 3 個の 8 段構成となっている。その他に鍵拡張の処理があるが、ハードウェアとしては、結線のみの構成で実現されるため primitive 構成の見積りとしては含んでいない。ダミー段処理 σ^0、中間鍵段関数 σ、中間鍵段関数 σ^{-1} の構成に関しても、ほぼ同程度の primitive 構成となっていることが判る。

拡大鍵生成部に関しては、F_σ 関数が 7 個、64 ビット排他的論理和（EXOR）が 35 個存在している。

鍵スケジュール部の場合は、データランダマイズ部と異なり、暗号化部と復号化部において、共通の鍵スケジュール部を利用することが可能である。

表 2. 鍵スケジュール部の各 primitive の見積り

中間鍵生成部	**ダミー段処理 σ^0**	**1 個**	**※暗号化･復号とも同じ構成**
	中間鍵段関数 σ	**4 個**	
	中間鍵段関数 σ^{-1}	**3 個**	

ダミー段処理 σ^0	**64 ビット EXOR**	**3 個**	
	M_{5E} 関数	**2 個**	
	F_σ 関数	**1 個**	
中間鍵段関数 σ 4 回実行	**64 ビット EXOR**	**3 個**	**※1 段辺り <u>256 ビット</u>の拡大鍵を生成**
	$P^{(32)}$ 関数	**1 個**	
	M_{5E} 関数	**2 個**	
	F_σ 関数	**1 個**	
中間鍵段関数 σ^{-1} 3 回実行	**64 ビット EXOR**	**3 個**	**※1 段辺り <u>384 ビット</u>の拡大鍵を生成**
	$P^{(32)-1}$ 関数	**1 個**	
	M_{B3} 関数	**1 個**	
	F_σ 関数	**1 個**	

F_σ 関数	**関数 S**	**8 個**
	$P^{(16)}$ 関数	**1 個**

拡大鍵生成部	**F_σ 関数**	**7 個**	**※暗号化･復号とも同じ構成**
	64 ビット EXOR	**35 個**	

５．ハードウェア評価結果

各 primitive の見積もり結果をもとにして、評価方針通りに評価を行った。

表 3 は、データランダマイズ部の各 primitive の実装結果である。ここで、ρ関数や XS 関数の実装に関しては、下位の primitive 毎に設計して積み上げる方式ではなく、実際に Verilog-HDL で記述し、論理合成を行った結果である。

表 3. データランダマイズ部の primitive 実装結果

	回路規模[Gate]	クリティカルパス[ns]
ρ関数	44895	12.47
XS 関数	43083	10.8
2to1selector*32bit	226	1.6
64 ビットレジスタ	515	0.80

表 4 は、鍵スケジュール部の各 primitive の実装結果である。中間鍵生成部に関しては、Feistel 構造であるため、ダミー段処理 σ^0 を 1 回、中間鍵段関数σを 4 回、中間鍵段関数 σ^{-1} を 3 回処理した結果の合計が、クリティカルパスとなる。それに対して、拡大鍵生成部では、それぞれを 1 段から 7 段までの処理をそれぞれ並列実行可能であることから、クリティカルパスは拡大鍵生成(平文側)の 1 回分である。

表 4. 鍵スケジュール部の primitive 実装結果

	回路規模[Gate]	クリティカルパス[ns]
32 ビット EXOR	267	1.08
中間鍵段関数σ	13422	7.43
ダミー段処理 σ^0	11741	6.57
中間鍵段関数 σ^{-1}	13136	5.42
中間鍵生成回路	83070	38.66
拡大鍵生成(平文側)	13186	5.76
拡大鍵生成(暗号文側)	12369	5.71

表 3、表 4 の primitive 実装結果をもとに、データランダマイズ部、鍵スケジュール部をアルゴリズムの構成を再現すると、表 5 のような結果となった。データランダマイズ部では暗号化部と復号化部で構成が異なるが、提示されている Hierocrypto-3 技術仕様書では、復号化回路に関する仕様が完全に記述されていない。そのため、正確な回路を構成することが困難であることから、本報告の評価では、暗号化回路を実装し、復号化回路に関しても、同等程度であるとの見積もりを行っている。

表 5. データランダマイズ部、鍵スケジュール部の評価結果

	回路規模 [Gate]	クリティカルパス[ns]
暗号化部	267558	73.15
復号化部	267558	73.15
2to1selector*128bit	904	1.6
64 ビットレジスタ*2	2059	0.80
小計（演算部分）	538078	75.55
中間鍵生成回路	83070	38.66
拡大鍵生成回路	89851	5.76
64 ビットレジスタ*26	13382	0.80
小計（鍵生成部分）	186302	45.22
合計	724380	75.55

以上の実装結果および見積もりにより、Hierocrypt-3 の H/W 詳細評価は、表 6 のような結果となった。なお、アルゴリズムの処理速度を見積もるため、本報告ではクリティカルパスに鍵スケジュールは含まれていないことに注意する。

表 6. Hierocrypt-3 の H/W 詳細評価結果

回路規模[Gate]	クリティカルパス[ns]	処理速度[Mbps]
724380	75.55	1694.24

６．まとめ

Hierocrypto-3 については、5 章の評価結果のように、約 1.7Gbps の処置速度であることから、128 ビット暗号として AES に採用された Rijndael 等と比較した場合、高速なアルゴリズムであると言える。しかし、回路規模に関しては、SPN 構造を採用しており、暗号化と復号化で共通の構成を採用できないため、約 653Kgate と非常に大きいアルゴリズムである。実際の回路実装を考慮した場合、ρ関数や中間鍵段関数σ等を全て実装することは無く、1 個実装し、ループして使用することが考えられるが、その場合においても、約 100～200Kgate 前後の規模となることが予想される。従って、IC カードや携帯端末などの小型なデバイスを使用するアプリケーションには向いていないアルゴリズムであると考えられる。

－　以上　－